University of Pennsylvania
Department of Electrical and System Engineering
System-on-a-Chip Architecture

ESE532, Fall 2019 — Midterm — Wednesday, October 9

- Exam ends at 11:50AM; begin as instructed (target 10:30AM)
  Do not open exam until instructed.
- Problems weighted as shown.
- Calculators allowed.
- Closed book = No text or notes allowed.
- Show work for partial credit consideration.
- Unless otherwise noted, answers to two significant figures are sufficient.
- Sign Code of Academic Integrity statement (see last page for code).

____

I certify that I have complied with the University of Pennsylvania's Code of Academic Integrity in completing this exam.

Name:

1	2a	2b	3	4	5	6	7	8	Total
10	5	5	10	10	10	10	20	20	100

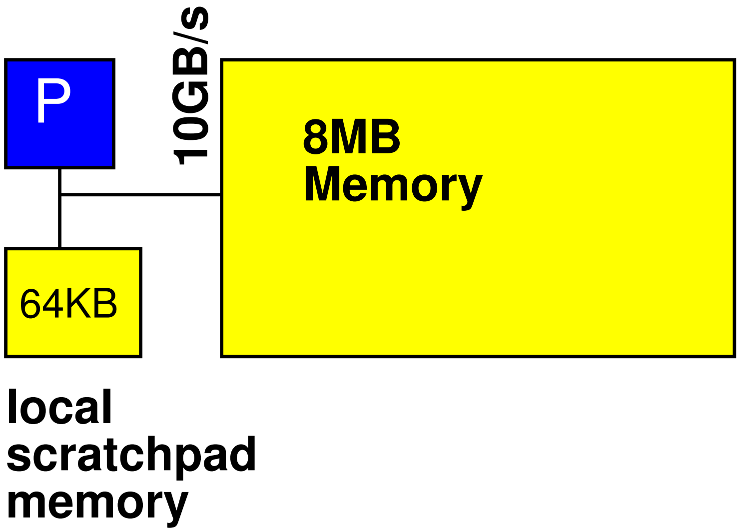
Consider the following code to pick a set of nearest-neighbor assignments (e.g., Ride Share Drivers to Passengers or Fire Trucks to Fires).

```
code_assign.c        Wed Oct 09 20:33:13 2019        1

void assign() {
    // these are all in the main memory
    uint32_t driver_x[AVAILABLE];
    uint32_t driver_y[AVAILABLE];
    uint16_t passenger_x[TARGETS];
    uint16_t passenger_y[TARGETS];
    uint64_t distances[TARGETS][AVAILABLE];
    uint16_t matches[TARGETS][TARGETS];
    uint16_t match[TARGETS];

    get_drivers(driver_x,driver_y); // in 10*AVAILABLE cycles all memory
    get_passengers(passenger_x,passenger_y); // in 10*TARGETS cycles all memory
    compute_distances(driver_x,driver_y,passenger_x,passenger_y,distances);
    best_matches(distances,matches);
    assign_matches(matches,match);
    //TYPO: send_assignments(match); // in 10*TARGET cycles all memory
    send_assignments(match); // in 10*TARGETS cycles all memory
  }
```

code_operators.c   Wed Oct 09 20:37:09 2019   1

```
#define TARGETS 100
#define AVAILABLE  1000

uint64_t distance(uint32_t x1, uint32_t y1, uint32_t x2, uint32_t y2) {
     int32_t dx=x1-x2;
     int32_t dy=y1-y2;
     return(dx*dx+dy*dy);
   }

void compute_distances(uint32_t driver_x[AVAILABLE],
                       uint32_t driver_y[AVAILABLE],
                       uint16_t passenger_x[TARGETS],
                       uint16_t passenger_y[TARGETS],
                       uint64_t distances[TARGETS][AVAILABLE]) {
    for (int p=0;p<TARGETS;p++) // loop A
       for (int d=0;d<AVAILABLE;d++) // loop B
          distances[p][d]=distance(driver_x[d],driver_y[d],
                                   passenger_x[p],passenger_y[p]);
    return;
   } // compute_distances

void best_matches(uint64_t distances[TARGETS][AVAILABLE],
                  uint16_t matches[TARGETS][TARGETS])      {

    uint64_t p_distances[AVAILABLE]; // in scratchpad memory
    uint16_t p_matches[TARGETS];     // in scratchpad memory
    for (int p=0;p<TARGETS;p++) {  // loop C
       for (int d=0;d<AVAILABLE;d++) // stream copy distances for p
          p_distances[d]=distances[p][d];
       closest_available(p_distances,p_matches);
       for (int m=0;m<TARGETS;m++) // stream copy matches for p
          matches[p][m]=p_matches[m];
       } // for p
    return;
} // best_matches

void closest_available(uint64_t distances[AVAILABLE],
                       uint16_t matches[TARGETS]) {
   // the matches result is an ordered list (smallest to largest)
   //  of the TARGETS nearest(smallest distance) available resources (drivers)

   // implementation omitted -- we will ask you to supply in Question 7.

   // for Question 1--5, assume
   // closest_available requires 4*TARGETS*AVAILABLE compute cycles
   //                        and 4*TARGETS*AVAILABLE memory cycles
   //                   critical path is TARGETS cycles (or, equivalently, ns)
}

void assign_matches(uint16_t matches[TARGETS][TARGETS], uint16_t match[TARGETS])
{
    // ERROR in exam: uint16_t driver_match[DRIVERS];  // in scratchpad memory
    uint16_t driver_match[AVAILABLE];  // in scratchpad memory
    for (int d=0;d<AVAILABLE;d++) driver_match[d]=0; // assume free
    for (int p=0;p<TARGETS;p++) // loop F
         for (int m=0;m<TARGETS;m++) // loop G
             if (driver_match[matches[p][m]]==0) {
                 match[p]=matches[p][m];
                 driver_match[matches[p][m]]=p;
                 break; // out of the for m loop
              } // if driver available
    return;
}
```

We start with a baseline, single processor system as shown.



- For simplicity throughout, we will treat non-memory indexing adds (subtracts count as adds), compares, and multplies as the only compute operations. We'll assume the other operations take negligible time or can be run in parallel (ILP) with the adds, multiplies, and memory operations. (Some consequences: You may ignore loop and conditional overheads in processor runtime estimates; you may ignore computations in array indecies.)
- Baseline processor can execute one multiply, compare, or add per cycle and runs at 1 GHz.
- Data can be transfered from the 8MB main memory at 10 GB/s when streamed in chunks of at least 192B. Assume `for` loops that only copy data can be auto converted into streaming operations.
- Non-streamed access to the main memory takes 10 cycles.
- Baseline processor has a local scratchpad memory that holds 64KB of data. Data can be streamed into the local scratchpad memory at 10 GB/s. Non-streamed accesses to the local scratchpad memory take 1 cycle.
- By default, all arrays live in the main memory.
- Arrays `p_distances`, `p_matches`, and `driver_match` live in local scratchpad memory.
- Assume scalar (non-array) variables can live in registers.
- Assume all additions are associative.
- Assume comparisons, adds, and multiplies take 1 ns when implemented in hardware accelerator, so fully pipelined accelerators also run at 1 GHz. A compare-mux operation can also be implemented in 1 ns.
- Data can be transfered to accelerator local memory at the same 10 GB/s when streamed in chunks of at least 256B.

1. Simple, Single Processor Resource Bounds

   Give the single processor resource bound time for each function.

function	Compute	Memory
compute_distances		
best_matches		
assign_matches		
assign		

2. Based on the simple, single processor mapping from Problem 1:

   (a) What function is the bottleneck? (circle one)

   compute_distances

   best_matches

   assign_matches

   (b) What is the Amdahl's Law speedup if you only accelerate the identified function?

3. Parallelism in Loops

   (a) Classify the following loops as data parallel or not? (loop bodies could be executed concurrently)

   (b) Explain why or why not?

Loop	Data Parallel?	Why or why not?
A		
B		
C		
F		
G		

4. Identify streaming opportunties between functions. When streaming is possible, what granularity of data can be usefully sent between the functions? [i.e., producer can generate and consumer can operate upon without waiting for additional data from the producer.] Report as a size in bytes and the logical data structure (or part of a data structure) to which this corresponds.

   (a) compute_distances → best_matches

   (b) best_matches → assign_matches

5. What is the critical path for the entire computation as captured in the `assign` function?

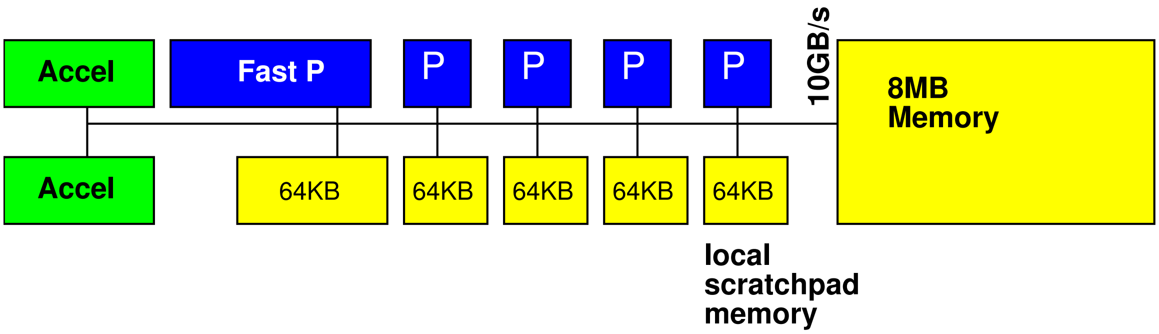
6. Rewrite the body of `compute_distances` to minimize the memory resource bound by exploiting the scratchpad memory.

   - Annotate what arrays live in the local scratchpad
   - Account for total memory usage in the local scratchpad
   - use for loops that only copy data to denote the streaming operations

   Estimate the new memory resource bound for your optimized `compute_distances`.

(This page intentionally left mostly blank for answers.)

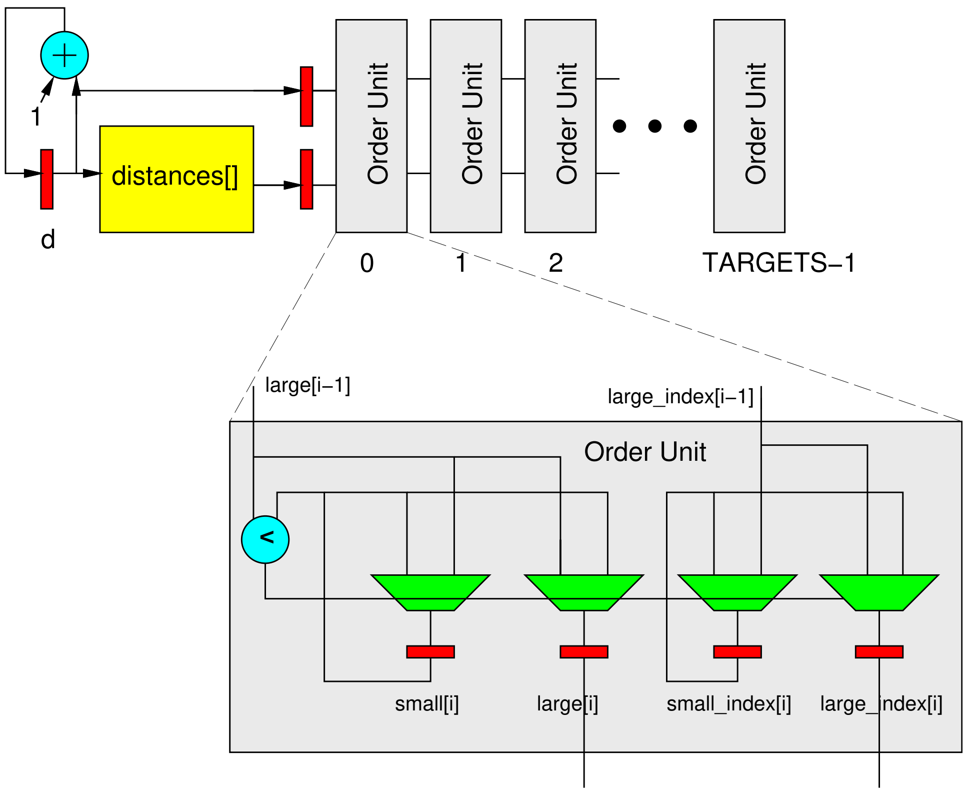
7. Consider a substrate with 4 simple processors (1 GHz as previously outlined), 1 fast processor (3 GHz, with everything running 3× as fast except data transfer from main memory), and 2 accelerators. The accelerators are pipelined and designed to start one call to `closest_available` every AVAILABLE cycles.[1]; pipeline depth is TARGETS. (You wlll look at the accelerator in Problem 8, but do not need to know its internal details to solve this problem). Describe how you would map the computation onto these heterogeneous computing resources. Describe how you would use the scratchpad memories as necessary beyond what you've already answered in Problem 6. Estimate the performance your mapping achieves.



[1]Original exam statement said "each cycle", which is not consistent with Problem 8.

(This page intentionally left mostly blank for answers.)

8. Consider the following accelerator for `closest_available` that processes one distance per cycle:



(Hint: this is performing an insertion sort one distance at a time, but keeping only the smallest TARGETS values.)

Assume:

- distances[] can be streamed from main memory into the local memory shown (with code already shown in best_matches)
- large[i] and small[i] can be reset to a maximum value, `MAX_VALUE`, in one cycle in hardware at the beginning of a call to `closest_available`; code for this reset in the C version is provided.
- small_index[i] and large_index[i] can be reset to a designated empty value, `EMPTY_VALUE`, in one cycle in hardware; code for this reset in the C version is provided.
- small_index[] becomes matches[] when the computation is completed and can be streamed into main memory (with code already shown in best_matches)

(a) Write C code that is consistent with the accelerator.

(b) Annotate the C code to explain how the C code was realized in the accelerator. (As appropriate, explain how unrolling, array partitioning, pipelining, dataflow streaming, and/or inlining were applied to the C code to get the implementation show.)

```
void closest_available(uint64_t distances[AVAILABLE],
                       uint16_t matches[TARGETS]) {

  // add declarations here (hint: what arrays do you need to declare?)




  for (int i=0;i<TARGETS;i++) small[i]=MAX_VALUE;
  for (int i=0;i<TARGETS;i++) small_index[i]=EMPTY_VALUE;
  for (int i=0;i<TARGETS;i++) large[i]=MAX_VALUE;
  for (int i=0;i<TARGETS;i++) large_index[i]=EMPTY_VALUE;

  // add code here (hint: what loop(s)?  what is the code for loop body (bodies?)




  for (int i=0;i<TARGETS;i++) matches[i]=small_index[i];
  return;
}
```

Code of Academic Integrity

Since the University is an academic community, its fundamental purpose is the pursuit of knowledge. Essential to the success of this educational mission is a commitment to the principles of academic integrity. Every member of the University community is responsible for upholding the highest standards of honesty at all times. Students, as members of the community, are also responsible for adhering to the principles and spirit of the following Code of Academic Integrity.*

Academic Dishonesty Definitions

Activities that have the effect or intention of interfering with education, pursuit of knowledge, or fair evaluation of a students performance are prohibited. Examples of such activities include but are not limited to the following definitions:

A. Cheating Using or attempting to use unauthorized assistance, material, or study aids in examinations or other academic work or preventing, or attempting to prevent, another from using authorized assistance, material, or study aids. Example: using a cheat sheet in a quiz or exam, altering a graded exam and resubmitting it for a better grade, etc.

B. Plagiarism Using the ideas, data, or language of another without specific or proper acknowledgment. Example: copying another persons paper, article, or computer work and submitting it for an assignment, cloning someone elses ideas without attribution, failing to use quotation marks where appropriate, etc.

C. Fabrication Submitting contrived or altered information in any academic exercise. Example: making up data for an experiment, fudging data, citing nonexistent articles, contriving sources, etc.

D. Multiple Submissions Multiple submissions: submitting, without prior permission, any work submitted to fulfill another academic requirement.

E. Misrepresentation of academic records Misrepresentation of academic records: misrepresenting or tampering with or attempting to tamper with any portion of a students transcripts or academic record, either before or after coming to the University of Pennsylvania. Example: forging a change of grade slip, tampering with computer records, falsifying academic information on ones resume, etc.

F. Facilitating Academic Dishonesty Knowingly helping or attempting to help another violate any provision of the Code. Example: working together on a take-home exam, etc.

G. Unfair Advantage Attempting to gain unauthorized advantage over fellow students in an academic exercise. Example: gaining or providing unauthorized access to examination materials, obstructing or interfering with another students efforts in an academic exercise, lying about a need for an extension for an exam or paper, continuing to write even when time is up during an exam, destroying or keeping library materials for ones own use., etc.

* If a student is unsure whether his action(s) constitute a violation of the Code of Academic Integrity, then it is that students responsibility to consult with the instructor to clarify any ambiguities.